UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/539,197	03/30/2000	Kelly Gene Johnson	03048.P011	8640

23363 7590 04/07/2004

CHRISTIE, PARKER & HALE, LLP
350 WEST COLORADO BOULEVARD
SUITE 500
PASADENA, CA 91105

EXAMINER	
WOOD, WILLIAM H	
ART UNIT	**PAPER NUMBER**
2124	17

DATE MAILED: 04/07/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/539,197	JOHNSON ET AL.	
	Examiner	Art Unit	
	William H. Wood	2124	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *23 January 2004*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-54* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-6,9-23 and 26-54* is/are rejected.

7)☒ Claim(s) *7,8,24 and 25* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

Claims 1-54 are pending and have been examined.

Claim Rejections - 35 USC § 102

1. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

2. Claims 13-15, 33-35, 40, 43, 46, 49, 52 and 54 are rejected under 35 U.S.C. 102(b) as being anticipated by Jonathan B. **Rosenberg**, "How Debuggers Work: Algorithms, Data Structures, and Architecture".

Claim 13

Rosenberg disclosed a method of debugging an executing service on a pipelined CPU architecture, the method comprising:

- setting a breakpoint at a last safe point location in an instruction set behind a first breakpoint location if the first breakpoint location is at an unsafe location in the instruction set *(page 113-116, Internal Breakpoints; specifically page 115, middle of third paragraph)*;
- saving a minimum state of the executing service *(page 99, Causing the Debuggee to Run, bottom)*;
- simulating instructions of the executing service from the last safe point to a next safe point past the breakpoint *(page 115, third paragraph)*;

- executing debug commands within the executing service *(page 99,* bottom of page; switching from debuggee (executing service) to debugger*)*; and
- restoring the state of the executing service *(page 99, Causing the Debuggee to Run, bottom)*.

Claim 14

Rosenberg disclosed the method of claim 13 wherein restoring further comprises:

- storing the simulated state of the executing server to the CPU *(page 99, Causing the Debuggee to Run, bottom; context switching/saving)*; and
- restoring an original execution *(page 99, Causing the Debuggee to Run, bottom; context switching/restoring)*.

Claim 15

Rosenberg disclosed the method of claim 13 wherein simulating further comprises single stepping through a set of unsafe instructions, the set of unsafe instructions are between the last safe point and a next safe point *(page 115, third paragraph, "single-step")*.

Claim 46

Rosenberg disclosed the method of claim 1, wherein saving the minimum state further comprises saving a minimum amount of the executing service that can be restored to halt and restart execution of the service without altering the behavior of the executing

service *(page 99, Causing the Debuggee to Run; context switch; minimum in order for processor to later resume, note 2 at bottom of page)*.

Claims 33-35, 40, 43, 49, 52 and 54

The claims contain limitations, which correspond to the limitations found in claims 13-15 and 46 above. As such, claims 33-35, 40, 43, 49, 52 and 54, are rejected the same manner as claims 13-15 and 46 above.

Claim Rejections - 35 USC § 103

3. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

4. Claims 1-6, 9, 10, 18-23, 26-27, 39, 42, 45, 48, 51 and 53 are rejected under 35 U.S.C. 103(a) as being unpatentable over Jonathan B. **Rosenberg**, "How Debuggers Work: Algorithms, Data Structures, and Architecture" in view of **Deao** et al. (USPN 6,112,298).

Claim 1

Rosenberg disclosed a method of debugging an executing service on a pipelined CPU architecture *(page 45-53, Contemporary CPU Debug Architectures)*, the method comprising:

- setting a breakpoint within an executing service *(page 98, Setting a Breakpoint)*;
- saving a minimum state of the executing service *(page 99, Causing the Debuggee to Run; context switch; minimum in order for processor to later resume, note 2 at bottom of page)*;
- setting a program counter of the executing service to point to a save stub *(page 99, context saving function of the OS; note at bottom of page)*;
- setting the program counter of the executing service to point to a restore stub *(page 99, context saving function of the OS; note at bottom of page)*; and
- restoring the state of the executing service *(page 99, context saving function of the OS; note at bottom of page)*.

Rosenberg did not explicitly state *executing one or more no-op instructions to flush the pipeline, if there is data in the pipeline that needs to be saved.* **Deao** demonstrated that it was known at the time of invention to utilize flushing a pipeline with no-ops in order to save state information (column 5, lines 24-36, lines 44-46, lines 50-54; column 49, lines 39-44, liens 66-67; column 50, lines 6-7, lines 15-35). It would have been obvious to one of ordinary skill in the art at the time of invention to implement the debugging system of **Rosenberg** with instruction pipeline flushing and saving as found in **Deao**'s teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide an accurate method of maintaining important hardware/pipeline process information and context, which is indicated as generally desirable by **Rosenberg** (page 99, lines 26-28).

Claim 2

Rosenberg disclosed the method of claim 1 further comprising:

- executing debug commands within the executing service *(page 99, bottom of page; switching from debuggee (executing service) to debugger)*.

Claim 3

Rosenberg disclosed the method of claim 1 wherein setting the breakpoint further comprises:

- locating an original instruction within the executing service to set the breakpoint *(page 108, Breakpoint Data structure; logical and physical breakpoints)*;
- inserting a breakpoint instruction at the breakpoint *(page 107-108; Breakpoints)*;
- starting the executing service *(page 99, Causing the Debuggee to Run; first sentence)*;
- waiting for the breakpoint to execute *(page 99, Causing the Debuggee to Run; first sentence)*;
- waiting for memory fetches and configuration loads to complete *(inherent to processors)*; and
- restoring the original instruction at the breakpoint location *(page 119-121; Single-Step)*.

Claim 4

Rosenberg disclosed the method of claim 1 wherein setting the breakpoint comprises:

- altering an instruction within the executing service at a breakpoint location *(page 98-99, Setting a Breakpoint; "Breakpoints are special instructions inserted into the executable ..."*

  , interrupt instructions inserted and altering the already present instructions);

Rosenberg did not explicitly state invalidating a page cache of the executing service. Official Notice is taken that it was known at the time of invention to invalidate page caches. It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg**'s system with invalidating a page cache of the debuggee (executing service). This implementation would have been obvious because one of ordinary skill in the art would be motivated to mark defective page caches as such (either they are mistakenly retrieved or containing errors).

Claim 5

Rosenberg disclosed the method of claim 1 wherein setting the breakpoint comprises:

- setting a breakpoint register to point to a breakpoint location *(page 46; fifth bulleted item)*.

Claim 6

Rosenberg disclosed the method of claim 1 wherein saving a minimum state comprises:

- saving the executing service registers *(page 99; near bottom; "... the operating system saves its stopped context (the values of all hardware registers ...")*;

Rosenberg did not explicitly state flushing a pipeline of the debuggee (executing service). Official Notice is taken that it was known at the time of invention to flush pipelines. It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosneberg**'s debugging systems with flushing a processor pipeline. This implementation would have been obvious because one of ordinary skill in the art would be motivated to prepare the pipeline for the new context as mentioned above (or in other words, remove the old context's values).

Claim 9

Rosenberg disclosed the method of claim 1 wherein [altering the program counter] further comprises:

- setting the program counter of the executing service to point to a save stub *(page 99-101, Causing the Debuggee to Run)*;
- starting execution of the executing service *(page 99-101, Causing the Debuggee to Run)*;
- executing the breakpoint *(page 99-101, Causing the Debuggee to Run)*;

- storing configuration registers of the executing service *(page 99-101, Causing the Debuggee to Run)*;
- saving values of registers *(page 99-101, Causing the Debuggee to Run)*;
- saving pipeline registers *(page 99-101, Causing the Debuggee to Run)*; and
- storing a stack pointer value for a breakpoint location *(page 99, saving context; page 136, The Program Stack, explains need for a program stack pointer and the stacks value to context)*.

Rosenberg did not explicitly state scalar and predicate registers. Official Notice is taken that it was known at the time of invention to make use of scalar and predicate registers. It would have been obvious to one of ordinary skill in the art at the time of invention to implement the processors of **Rosenberg** with scalar and predicate registers. This implementation would have been obvious because one of ordinary skill in the art would be motivated to implement microprocessors using commonly understood technology such as scalar and predicate registers.

Claim 10

Rosenberg disclosed the method of claim 1 restoring the program counter further comprising:

- starting the executing service at the breakpoint *(page 99, note 2 at bottom of page)*.

Claim 22

Rosenberg disclosed the system of claim 18 wherein the save stub is further operable to save the executing service registers *(page 99; near bottom; "... the operating system saves its stopped context (the values of all hardware registers ...")*.

Claim 45

Rosenberg disclosed the method of claim 1, wherein saving the minimum state further comprises saving a minimum amount of the executing service that can be restored to halt and restart execution of the service without altering the behavior of the executing service *(see above for claim 1 on minimum)*.

Claims 18-19, 21-22, 27, 39, 42, 48, 51 and 53

The claims contain limitations, which correspond to the limitations found in claims 1-3, 5, 10, 13-15 and 45 above. As such, claims 18-19, 21-22, 27, 33-35, 39-40, 42-43, 46, 48-49 and 51-54, are rejected the same manner as claims 1-3, 5, 10, 13-15 and 45 above.

Claims 20, 23, 26

The claims contain limitations, which correspond to the limitations found in claims 4, 6 and 9 above. As such, claims 20, 23, 26, are rejected the same manner as claims 4, 6 and 9 above.

5. Claims 11 and 28-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Jonathan B. **Rosenberg**, "How Debuggers Work: Algorithms, Data Structures, and Architecture" in view of **Deao** et al. (USPN 6,112,298) and in further view of **Wu** (USPN 5,404,428).

Claim 11

Rosenberg disclosed the method of claim 1 wherein restoring the state further comprises:

- if a breakpoint location is on an instruction that does not make use of old values, restoring stable registers *(page 99; all registers restored, regardless of whether the value is done in the pipeline or not)*;
- if the breakpoint location is on an instruction that does make use of old values, restoring unstable registers *(page 99; all registers restored, including the pipeline context)*;
- altering the program counter of the executing service to point to the breakpoint location *(page 99)*; and
- starting execution of the executing service at the breakpoint location *(page 99)*.

Rosenberg did not explicitly state reloading pipeline. **Wu** demonstrated that it was known at the time of invention to reload a software pipeline to restore state (column 16, lines 8-9). It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg**'s various hardware pipelines with the ability to be

reloaded as found in **Wu**'s teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to prepare a pipeline to return to the previous task as before the context switch (illustrated by **Rosenberg**).

Claim 28

As the limitations of claim 28 correspond to claim 11, claim 28 is rejected in the same manner.

Claim 29

Rosenberg and **Wu** did not explicitly state the system of claim 28 wherein the restore stub is further operable to reload the pipeline state *directly*. Official Notice is taken that it was known at the time of invention to reload directly. It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg** and **Wu** with direct reloading. This implementation would have been obvious because one of ordinary skill in the art would be motivated to reload the pipeline as quickly as possible, furthermore **Rosenberg** mentions restoring registers, which would imply direct reloading.

Claim 30

Rosenberg and **Wu** did not explicitly state the system of claim 28 wherein the restore stub is further operable to re-execute the original instructions within the pipeline to recreate the pipeline at a time of the breakpoint. Official Notice is taken that it was

known at the time of invention to recreate the pipeline state via re-executing original instructions. It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg** and **Wu** with re-execution and recreation of state. This implementation would have been obvious because one of ordinary skill in the art would be motivated to reload the pipeline as quickly as possible and store as little information as possible (both aid pipeline resources), therefore simply resuming execution at an instruction would be highly efficient in terms of needed memory and transfer bandwidth.

6. Claims 12, 16-17, 31-32, 36-38, 41, 44, 47 and 50 are rejected under 35 U.S.C. 103(a) as being unpatentable over Jonathan B. **Rosenberg**, "How Debuggers Work: Algorithms, Data Structures, and Architecture" in view of **Deao** et al. (USPN 6,112,298) and in further view of "Dictionary of Computing" herein referred to as **Computing**.

Claim 12

Rosenberg disclosed the method of claim 1 further comprising:

- fetching a page of memory of the executing service into an instruction cache *(page 45-53, included in processors mentioned x86, PowerPC, etc.)*;

Rosenberg did not explicitly state checking for a checksum error within the page of memory. **Computing** demonstrated that it was known at the time of invention to make use of checksum's to determine errors (page 72). It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg**'s debugging

processors with checksum as found in **Computing**'s teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to utilize a "simple" error detection mechanism to provide accurate data.

Rosenberg did not explicitly state if the executing service is set to reject the checksum error, saving the page of memory, inserting a breakpoint into the saved page of memory, altering an instruction pointer to the saved page of memory, and processing the saved page of memory. **Computing** demonstrated that it was known at the time of invention to make use of checksum's to determine errors (page 72). It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg**'s debugging processors with checksum as found in **Computing**'s teaching and thus debugging a page which contained errors. This implementation would have been obvious because one of ordinary skill in the art would be motivated to utilize a "simple" error detection mechanism to provide accurate data and the debug capability of **Rosenberg** to ensure possible faulty data can be executed correctly or find the error, thus allowing the processor continued uninterrupted processing (which is important to real-time systems).

Claim 16

Rosenberg disclosed a method of debugging an executing service on a pipelined CPU architecture without hardware interlocks *(page 45-53, Contemporary CPU Debug Architectures)*, the method comprising:

- fetching a page of memory of the executing service into an instruction cache *(included in processors mentioned x86, PowerPC, etc.)*;
- setting a breakpoint within an executing service *(page 98, Setting a Breakpoint)*;

Rosenberg did not explicitly state checking for a checksum error within the page of memory. **Computing** demonstrated that it was known at the time of invention to make use of checksum's to determine errors (page 72). It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg**'s debugging processors with checksum as found in **Computing**'s teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to utilize a "simple" error detection mechanism to provide accurate data.

Rosenberg did not explicitly state if the executing service is set to reject the checksum error, saving the page of memory, inserting a breakpoint into the saved page of memory, altering an instruction pointer to the saved page of memory, and processing the saved page of memory. **Computing** demonstrated that it was known at the time of invention to make use of checksum's to determine errors (page 72). It would have been obvious to one of ordinary skill in the art at the time of invention to implement **Rosenberg**'s debugging processors with checksum as found in **Computing**'s teaching and thus debugging a page which contained errors. This implementation would have been obvious because one of ordinary skill in the art would be motivated to utilize a "simple" error detection mechanism to provide accurate data and the debug capability of

Rosenberg to ensure possible faulty data can be executed correctly or find the error, thus allowing the processor continued uninterrupted processing (which is important to real-time systems).

Rosenberg did not explicitly state *executing one or more no-op instructions to flush the pipeline, if there is data in the pipeline that needs to be saved.* **Deao** demonstrated that it was known at the time of invention to utilize flushing a pipeline with no-ops in order to save state information (column 5, lines 24-36, lines 44-46, lines 50-54; column 49, lines 39-44, liens 66-67; column 50, lines 6-7, lines 15-35). It would have been obvious to one of ordinary skill in the art at the time of invention to implement the debugging system of **Rosenberg** with instruction pipeline flushing and saving as found in **Deao**'s teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide an accurate method of maintaining important hardware/pipeline process information and context, which is indicated as generally desirable by **Rosenberg** (page 99, lines 26-28).

Claim 17

Rosenberg and **Computing** disclosed the method of claim 16 wherein processing further comprises:

- setting a breakpoint within an executing service;
- saving a minimum state of the executing service;
- altering a program counter of the executing service;

- executing debug commands within the executing service;
- restoring the program counter of the executing service; and
- restoring the state of the executing service.

All disclosed as shown above by **Rosenberg** (for example, page 99-101).

Claims 31-32, 36-38, 41 and 44

The claims contain limitations, which correspond to the limitations found in claims 12 and 16-17 above. As such, claims 31-32, 36-38, 41 and 44, are rejected the same manner as claims 12 and 16-17 above.

Claims 47 and 50

The claims contain limitations, which correspond to the limitations found in claim 45 above. As such, claims 47 and 50, are rejected the same manner as claim 45 above.

Allowable Subject Matter

7. Claims 7, 8, 24 and 25 are allowed. The prior art of record does not teach or fairly suggest the claim limitations of claims 7, 8, 24 and 25.

Response to Arguments

8. Applicant's arguments filed 23 January 2004 have been fully considered but they are not persuasive. Applicant argued **Rosenberg** did not disclose *setting a breakpoint at a last safe point location in an instruction set in the pipeline behind a first breakpoint*

location if the first breakpoint location is at an unsafe location in the pipeline. This is incorrect. **Rosenberg** did, in actuality, disclose the above limitation as stated in the previous rejection. Setting a breakpoint at the target meets the requirements of the limitation by [1)] being behind the branch instruction and [2)] being a safe location, the branch itself not being safe. Applicant appears to have argued the target is not the not the next instruction after the branch (Remarks received 23 January 2004, page 21, lines 1-4). This is also incorrect. By definition of branching, the target is the next instruction. Furthermore, **Rosenberg** last sentence of the third paragraph on page 115 states the debugger could optionally place breakpoints at either target (indicating the very next instruction no matter what happens at the branch). Therefore, **Rosenberg** disclosed the above limitation as claimed. This is believed to address all of Applicant's concerns over the rejected claim limitations.

Conclusion

9. This action is *not* made final. Claims 41 and 44, currently rejected, were not clearly rejected in the previous Office Action.

Correspondence Information

Any inquiry concerning this communication or earlier communications from the examiner should be directed to William H. Wood whose telephone number is (703)305-3305. The examiner can normally be reached 7:30am - 5:00pm Monday thru Thursday and 7:30am - 4:00pm every other Friday.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (703)305-9662. The fax phone numbers for the organization where this application or proceeding is assigned are (703)746-7239 for regular communications and (703)746-7238 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is (703)305-3900.

William H. Wood
March 24, 2004

KAKALI CHAKI
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100